baySeq: Empirical Bayesian analysis of patterns of differential expression in count data

Thomas J. Hardcastle

October 18, 2010

1 Introduction

This vignette is intended to give a rapid introduction to the commands used in implementing two methods of evaluating differential expression in Solexa-type, or *count* data by means of the `baySeq` R package. For fuller details on the methods being used, consult Hardcastle & Kelly [1]. The major improvement made in this release is the option to include region length in evaluating differential expression between genomic regions (e.g. genes). See Section 7 for more details.

We assume that we have discrete data from a set of sequencing or other high-throughput experiments, arranged in a matrix such that each column describes a sample and each row describes some entity for which counts exist. For example, the rows may correspond to the different sequences observed in a sequencing experiment. The data then consists of the number of times each sequence is observed in each sample. We wish to determine which, if any, rows of the data correspond to some patterns of differential expression across the samples. This problem has been addressed for pairwise differential expression by the `edgeR` [2] package.

However, `baySeq` takes an alternative approach to analysis that allows more complicated patterns of differential expression than simple pairwise comparison, and thus is able to cope with more complex experimental designs. We also observe that the methods implemented in `baySeq` perform at least as well, and in some circumstances considerably better than those implemented in `edgeR` [1].

`baySeq` uses empirical Bayesian methods to estimate the posterior likelihoods of each of a set of models that define patterns of differential expression for each row. This approach begins by considering a distribution for the row defined by a set of underlying parameters for which some prior distribution exists. By estimating this prior distribution from the data, we are able to assess, for a given model about the relatedness of our underlying parameters for multiple libraries, the posterior likelihood of the model.

In forming a set of models upon the data, we consider which patterns are biologically likely to occur in the data. For example, suppose we have count data from some organism in condition A and condition B. Suppose further that we have two biological replicates for each condition, and hence four libraries A_1, A_2, B_1, B_2, where A_1, A_2 and B_1, B_2 are the replicates. It is reasonable to suppose that at least some of the rows may be unaffected by our experimental conditions A and B, and the count data for each sample in these rows will be *equivalent*. These data need not in general be identical across each sample

due to random effects and different library sizes, but they will share the same underlying parameters. However, some of the rows may be influenced by the different experimental conditions A and B. The count data for the samples A_1 and A_2 will then be equivalent, as will the count data for the samples B_1 and B_2. However, the count data between samples A_1, A_2, B_1, B_2 will not be equivalent. For such a row, the data from samples A_1 and A_2 will then share the same set of underlying parameters, the data from samples B_1 and B_2 will share the same set of underlying parameters, but, crucially, the two sets will not be identical.

Our task is thus to determine the posterior likelihood of each model for each row of the data. We can do this by considering either a Poisson or negative-binomial distribution upon the sequencing count data. The Possion method is considerably faster as a closed form conjugate prior exists for this distribution. The negative-binomial solution is slower as it requires a numerical solution for the prior, but is probably a better fit for most data. In experimental data, we have found that the Poisson method is likely to give poor results if true biological replicates are not available; in most human studies, for example. In general, therefore, the use of the negative-binomial methods is recommended.

2 Preparation

We begin by loading the `baySeq` package.

```
> library(baySeq)
```

Note that because the experiments that `baySeq` is designed to analyse are usually massive, we should use (if possible) parallel processing as implemented by the `snow` package. We therefore need to load the `snow` package (if it exists), define a *cluster* and load the `baySeq` library onto each member of the cluster.

```
> library(snow)
> cl <- makeCluster(4, "SOCK")
```

If `snow` is not present, we can proceed anyway with a `NULL` cluster. Results may be slightly different depending on whether or not a cluster is used owing to the non-deterministic elements of the method.

```
> cl <- NULL
```

Here we have (if the `snow` package is installed) defined a cluster of four processors on sockets; that is to say, on the local machine. If the local machine has multiple processors this may be a valid method of accelerating `baySeq`, but if very large data sets are being analysed we may wish to consider some other form of parallelisation (e.g. LAM/MPI) that allows processors on multiple nodes to be used. See the `snow` documentation for details on how to achieve this.

We load a simulated data set consisting of count data on one thousand counts and library sizes for ten libraries.

```
> data(simCount)
> data(libsizes)
> simCount[1:10, ]
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    4    1    5    2    3    0    1    1    1     0
 [2,]    1    0    9    6    5    0    1    0    0     1
 [3,]    9    2    5    5   14    2    3    1    0     4
 [4,]    7    3    8    2    2    0    1    0    1     0
 [5,]    2    2    4    7    0    0    0    0    0     1
 [6,]    2    1    0    1    0    4    3    5    5     3
 [7,]    9    8    8    8    9    1    2    1    0     0
 [8,]    9    5    7    8    7    1    2    0    1     2
 [9,]    6    2    2    3    0    0    0    0    0     0
[10,]    1    0    2    0    1    3   17    2    2    10
```

```
> libsizes
```

```
 [1] 75373 40153 75403 34285 55975 53287 80477 37655 41171 77510
```

The data are simulated such that the first hundred counts show differential expression between the first five libraries and the second five libraries. Our replicate structure, used to estimate the prior distributions on the data, can thus be defined as

```
> replicates <- c(1, 1, 1, 1, 1, 2, 2, 2, 2, 2)
```

We can also establish two group structures for the data.

Each member (vector) contained within the 'groups' list corresponds to one model upon the data. In this setting, a model describes the patterns of data we expect to see at least some of the tags correspond to. In this simple example, we expect that some of the tags will be equivalently expressed between all ten libraries. This corresponds to the 'NDE' model, or vector `c(1,1,1,1,1,1,1,1,1,1)` - all libraries belong to the same group for these tags.

We also expect that some tags will show differential expression between the first five libraries and the second five libraries. For these tags, the two sets of libraries belong to different groups, and so we have the model 'DE', or vector `c(1,1,1,1,1,2,2,2,2,2)` - the first five libraries belong to group 1 and the second five libraries to group 2. We thus have the following group structure

```
> groups <- list(NDE = c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1), DE = c(1,
+     1, 1, 1, 1, 2, 2, 2, 2, 2))
```

In a more complex experimental design (Section 1) we might have several additional models. The key to constructing vectors corresponding to a model is to see for which groups of libraries we expect equivalent expression of tags.

We note that the group for DE corresponds to the replicate structure. This will often be the case, but need not be in more complex experimental designs.

The ultimate aim of the `baySeq` package is to evaluate posterior likelihoods of each model for each row of the data.

We begin by combining the count data, library sizes and user-defined groups into a `countData` object.

```
> CD <- new("countData", data = simCount, replicates = replicates,
+     libsizes = as.integer(libsizes), groups = groups)
```

We can then plot the data in the form of an MA-plot, suitable modified to plot those data where the data are uniformly zero (and hence the log-ratio is infinite) (Figure 1). Truly differentially expressed data can be identified in the plot by coloring these data red, while non-differentially expressed data are colored black.

```
> plotMA.CD(CD, samplesA = 1:5, samplesB = 6:10, col = c(rep("red",
+     100), rep("black", 900)))
```

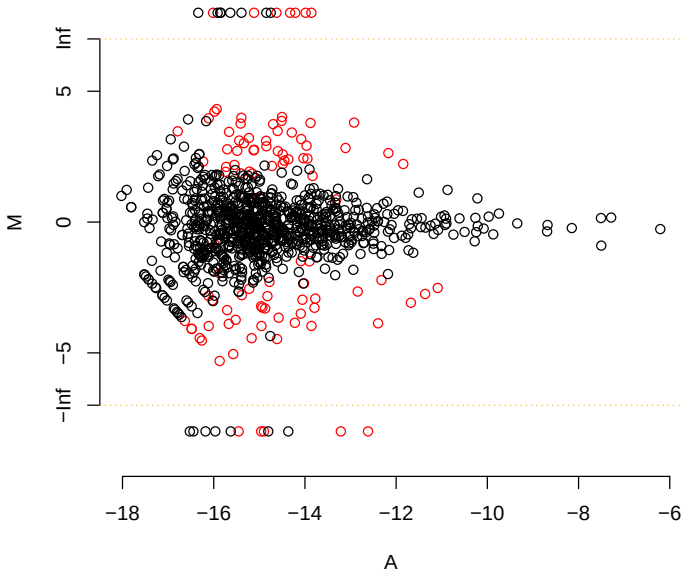

Figure 1: 'MA'-plot for count data. Where the log-ratio would be infinite (because the data in one of the sample groups consists entirely of zeros, we plot instead the log-values of the other group. Truly differentially expressed data are colored red, and non-differentially expressed data black.

We can also optionally add annotation details into the `@annotation` slot of the `countData` object.

```
> CD@annotation <- data.frame(name = paste("count", 1:1000, sep = "_"))
```

3 Poisson-Gamma Approach

We first try to identify the posterior likelihoods of each model for each tag assuming a Poisson distribution on each tag with a rate that is Gamma distributed. That is, if Y_{ij} is an element of the data where i is the row of the data,

and j is the sample, then

$$Y_{ij} \sim Poi(\theta_j l_j)$$

where the l_j is the library size of sample j (or some other suitable scaling factor) and

$$\theta_j \sim \Gamma(\alpha_j, \beta_j)$$

The relationships between the α_j, β_j for each j are determined by the model being investigated such that, if and only if samples X and Y belong to the same group, then $\alpha_X = \alpha_Y$ and $\beta_X = \beta_Y$.

We begin by trying to establish the parameters of the Gamma distribution by bootstrapping from the data and applying maximum likelihood methods. We are able to adjust the parameters of the bootstrapping; here we take twenty sets of count data, establish the maximum likelihood Gamma parameters, and iterate over 1000 cases. In general more than 5000 iterations is recommended but is used here for speed of calculation.

We then take the mean of the maximum likelihood estimates to acquire a prior on the rate distribution.

```
> CDP.Poi <- getPriors.Pois(CD, samplesize = 20, takemean = TRUE,
+     cl = cl)
```

The calculated priors are stored in the `@priors` slot of the `countData` object produced.

```
> CDP.Poi@priors

$priors
$priors$NDE
$priors$NDE[[1]]
[1]     1.086451 21304.408980


$priors$DE
$priors$DE[[1]]
[1]     0.890923 16209.662922

$priors$DE[[2]]
[1]     0.965833 18823.746933
```

For each model, we get a set of priors. In the Poisson-Gamma approch, we get, for each group in the model, a pair of parameters which define the Gamma distribution that we shall use as a prior distribution for the rates of the Poisson distributions that describe how many counts we see in each row of the data. Thus, in the model of differential expression, there are two groups in the data and we find two sets of parameters.

Having acquired a set of prior distributions on the rate parameter of the Poisson distribution, we can calculate the posterior likelihoods of each model for each tag. We need to either provide an initial prior likelihood on each model via the `prs` parameter, or provide some other means of estimating the prior

likelihood on the model. If 'pET = "BIC" then the prior likelihood on each model will be estimated from the Bayesian Information Criterion, and the 'prs' parameter is not necessary (and will be ignored).

```
> CDPost.Poi <- getLikelihoods.Pois(CDP.Poi, pET = "BIC", cl = cl)
> CDPost.Poi@estProps

      NDE        DE
0.7522801 0.2477199

> CDPost.Poi@posteriors[1:10, ]

              NDE            DE
 [1,]  -1.886105 -1.644754e-01
 [2,]  -6.334655 -1.775332e-03
 [3,]  -4.694672 -9.185931e-03
 [4,]  -6.927718 -9.807157e-04
 [5,]  -4.401511 -1.233456e-02
 [6,]  -2.761509 -6.528153e-02
 [7,] -15.491184 -1.871821e-07
 [8,]  -9.199031 -1.011425e-04
 [9,]  -5.204166 -5.508777e-03
[10,]  -9.821010 -5.430017e-05

> CDPost.Poi@posteriors[101:110, ]

                NDE            DE
 [1,] -0.1112734239  -2.250885686
 [2,]  0.0000000000 -29.862948332
 [3,] -0.0775453936  -2.595413946
 [4,] -0.0600011359  -2.843242353
 [5,] -0.0009654491  -6.943399847
 [6,] -0.0855247885  -2.501406664
 [7,] -0.1857843054  -1.774623334
 [8,] -5.7272248308  -0.003261414
 [9,] -0.0789094037  -2.578650143
[10,] -0.0491365691  -3.037619420
```

The estimated posterior likelihoods for each model are stored in the natural logarithmic scale in the `@posteriors` slot of the `countDataPosterior`. The nth column of the posterior likelihoods matrix corresponds to the nth model as listed in the `group` slot of `CDPost.Poi`.

Here the assumption of a Poisson distribution gives an estimate of

```
      DE
0.2477199
```

as the proportion of differential expressed counts in the simulated data, where in fact the proportion is known to be 0.1.

4 Negative-Binomial Approach

We next try the same analysis assuming a Negative Binomial distribution on the data. We first estimate an empirical distribution on the parameters of the Negative Binomial distribution by bootstrapping from the data, taking individual counts and finding the quasi-likelihood parameters for a Negative Binomial distribution. By taking a sufficiently large sample, an empirical distribution on the parameters is estimated. A sample size of around 10000 iterations is suggested, depending on the data being used), but 1000 is used here to rapidly generate the plots and tables.

```
> CDP.NBML <- getPriors.NB(CD, samplesize = 1000, estimation = "QL",
+     cl = cl)
```

The calculated priors are stored in the `@priors` slot of the `countData` object produced as before. For the negative-binomial method, we are unable to form a conjugate prior distribution. Instead, we build an empirical prior distribution which we record in the list object `$priors` of the slot `@priors`. Each member of this list object corresponds to one of the models defined by the `group` slot of the `countData` object and contains the estimated parameters for each of the individual counts selected under the models. The vector `$sampled` contained in the slot `@priors` describes which rows were sampled to create these sets of parameters.

We then acquire posterior likelihoods as before, estimating the proportions of differentially expressed counts. We can repeatedly bootstrap the prior estimatation to improve accuracy; here three bootstraps are used.

```
> CDPost.NBML <- getLikelihoods.NB(CDP.NBML, pET = "BIC", cl = cl)
```

.

```
> CDPost.NBML@estProps

[1] 0.8741814 0.1258186

> CDPost.NBML@posteriors[1:10, ]

              NDE           DE
 [1,] -0.7474096 -0.641678252
 [2,] -1.0138774 -0.450686586
 [3,] -0.9121756 -0.513578496
 [4,] -2.5177825 -0.084075573
 [5,] -0.6836802 -0.702704593
 [6,] -0.9926478 -0.462978941
 [7,] -5.3744828 -0.004644081
 [8,] -4.1496960 -0.015894866
 [9,] -1.1577209 -0.377171367
[10,] -1.7246166 -0.196308578

> CDPost.NBML@posteriors[101:110, ]
```

```
                NDE        DE
 [1,] -5.726230e-02 -2.888607
 [2,] -6.435923e-05 -9.651062
 [3,] -4.918016e-02 -3.036754
 [4,] -1.618569e-02 -4.131710
 [5,] -5.464533e-03 -5.212208
 [6,] -5.141471e-02 -2.993428
 [7,] -7.869146e-02 -2.581308
 [8,] -4.906630e-02 -3.039016
 [9,] -6.350955e-02 -2.788152
[10,] -9.929686e-03 -4.617187
```

Here the assumption of a Negative Binomial distribution with priors estimated by maximum likelihood gives an estimate of

```
[1] 0.1258186
```

as the proportion of differential expressed counts in the simulated data, where in fact the proportion is known to be 0.1.

5 Results

We can ask for the top differentially expressed tags using the `topCounts` function.

```
> topCounts(CDPost.NBML, group = 2)
```

```
       name X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 Likelihood
1  count_80  1  1  0  1  1 13 21  8  6  20  0.9989589
2  count_78  1  1  0  1  1  8 13  7  9  10  0.9980143
3  count_66  0  0  0  0  0 15 10  4  4  10  0.9972105
4  count_26 13  4 11  5  7  1  1  1  0   0  0.9969941
5  count_21  2  0  1  1  0 15 15  6  5  11  0.9962287
6   count_7  9  8  8  8  9  1  2  1  0   0  0.9953667
7  count_72  0  0  1  0  0  7  6  4  3   8  0.9927959
8  count_83 14  6  9  2  9  1  0  0  1   1  0.9925575
9  count_64  6  6  8 11  9  1  1  0  0   1  0.9907816
10 count_27  5  3  6  4  7  0  0  0  1   0  0.9857751
```

We can compare the accuracy of the methods by using the `getTPs` function to find the number of true positives. We then use this to plot the false positive rate.

```
> NBML.TPs <- getTPs(CDPost.NBML, group = 2, TPs = 1:100)
> Poi.TPs <- getTPs(CDPost.Poi, group = 2, TPs = 1:100)
```

We can plot the posterior likelihoods against the log-ratios of the two sets of samples using the `plotPosteriors` function, coloring the truly differentially expressed data red and the non-differentially expressed data black (Figure 2).

```
> plotPosteriors(CDPost.NBML, group = 2, samplesA = 1:5, samplesB = 6:10,
+     col = c(rep("red", 100), rep("black", 900)))
```

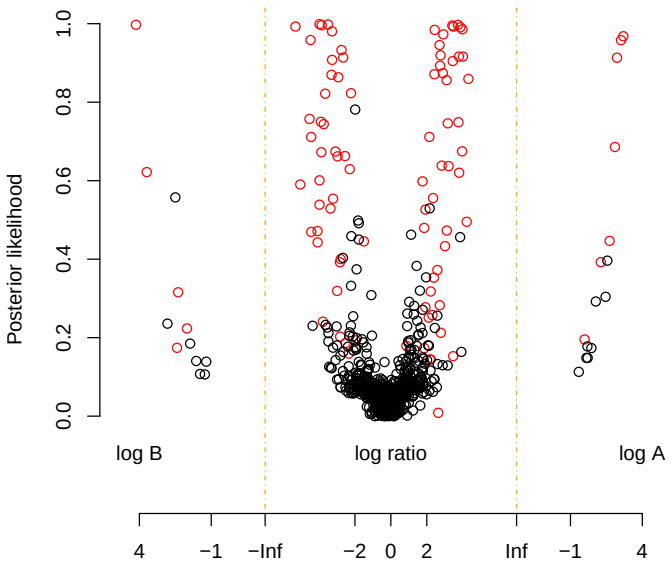

Figure 2: Posterior likelihoods of differential expression against log-ratio (where this would be non-infinite) or log values (where all data in the other sample group consists of zeros). Truly differentially expressed data are colored red, and non-differentially expressed data black.

Figure 3 shows that false positive rates for the bootstrapped Negative Binomial approach with maximum likelihood priors (red) are much lower than for the Poisson-Gamma conjugacy approach (blue). This approach is therefore significantly more accurate, although potentially computationally more intensive and thus slower than the Poisson-Gamma conjugacy.

Finally, we shut down the cluster (assuming it was started to begin with).

```
> if (!is.null(cl)) stopCluster(cl)
```

6 More Complex Experimental Designs

To illustrate the way in which a model is specified for more complex experimental designs, we consider a factorial design containing eight libraries. Table 1 describes the experimental design in more detail. Samples 1, 2, 3 & 4 are from condition A while samples 5, 6, 7 & 8 are from condition B. However, samples 1, 2, 5 & 6 have also been subjected to some condition C, while samples 3, 4, 7 & 8 have been subjected to some condition D.

We load a simulated data set corresponding to the factorial design described. The first hundred cases show differential expression caused by differences be-

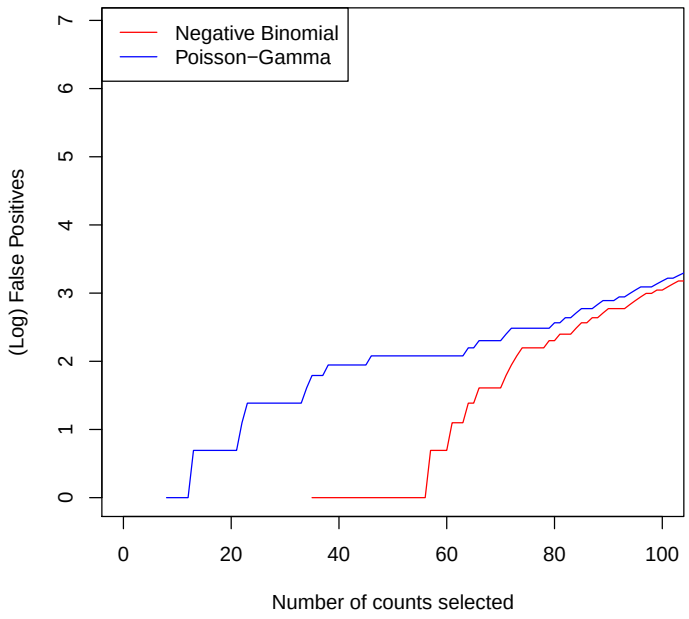

Figure 3: (Log) false positive rates for differentially expressed tag discovery in a simulated dataset using the Poisson-Gamma approach (blue) and the Negative Binomial approach (red).

tween condition A and condition B, while the second hundred cases show differential expression caused by differences between condition C and condition D.

```
> data(factData)
> data(factlibsizes)
```

We establish our replicate structure, together with three group structures on the data. We observe that in this case, the replicate structure does not correspond to any of the group structures.

```
> replicates <- c(1, 1, 2, 2, 3, 3, 4, 4)
> factgroups <- list(NDE = c(1, 1, 1, 1, 1, 1, 1, 1), DE.A.B = c(1,
+     1, 1, 1, 2, 2, 2, 2), DE.C.D = c(1, 1, 2, 2, 1, 1, 2, 2))
```

The first group assumes no differential expression between samples. The second group assumes differential expression between samples experiencing condition A and samples experiencing condition B. The third group assumes differential expression between samples experiencing condition C and samples experiencing condition D.

We could also consider the possibility of interactions between effects, by considering a group `c(1,1,2,2,3,3,4,4)`. However, in this simulated data set,

	Condition A	Condition B
Condition C	Samples 1, 2	Samples 5, 6
Condition D	Samples 3, 4	Samples 7, 8

Table 1: An example factorial design experiment in which samples 1 and 2 are subjected to experimental conditions A and C, samples 3 and 4 are subjected to conditions B and C, samples 5 and 6 are subjected to conditions A and C and samples 7 and 8 are subjected to conditions B and D.

no such data exists and so we need not consider this group. It should be noted, however, that such a group would only find that an interaction effect takes place in some elements of the data. Like an ANOVA test, it is necessary to examine the data to determine what form the effect takes.

Having established a group structure, we proceed as before.

```
> CDfact <- new("countData", data = factCount, replicates = replicates,
+     libsizes = factlibsizes, groups = factgroups)
> CDfact@annotation <- data.frame(name = paste("count", 1:1000,
+     sep = "_"))
> CDfactP.NBML <- getPriors.NB(CDfact, samplesize = 1000, estimation = "QL",
+     cl = cl)
> CDfactPost.NBML <- getLikelihoods.NB(CDfactP.NBML, pET = "BIC",
+     cl = cl)

.

> CDfactPost.NBML@estProps

[1] 0.6875271 0.1631200 0.1493529
```

We can then ask for the tags showing most differential expression caused by the difference between conditions A and B

```
> topCounts(CDfactPost.NBML, group = 2)

       name X1 X2 X3 X4 X5 X6 X7 X8 Likelihood
1  count_94 19 41 19 22  3  2  3  4  0.9997025
2  count_81  6 12  5 10  0  0  0  0  0.9985424
3  count_88  0  0  0  0 10  8  6  8  0.9982633
4  count_41  2  5  4  1 29 11 29 23  0.9979602
5  count_13  2  6  6  6 32 32 59 37  0.9974925
6  count_60  5 14  6  6  0  0  1  0  0.9974013
7  count_49  5 19 14 12  2  1  2  1  0.9955937
8   count_7  0  1  1  0 19 16  9  8  0.9952682
9  count_75  1  4  1  0 30  7 33 14  0.9926413
10 count_95  3 14 13  8  1  0  0  0  0.9925988
```

And for those tags showing most differential expression caused by the difference between conditions C and D

```
> topCounts(CDfactPost.NBML, group = 3)
```

```
        name X1 X2 X3 X4 X5 X6 X7 X8 Likelihood
1  count_138  3  1 21 23  2  1 29 26  0.9998942
2  count_126 53 78 10  3 47 39  7  6  0.9992812
3  count_161 15 27  1  1 10 15  1  0  0.9991646
4  count_200 18 16  0  1 11 10  1  2  0.9980369
5  count_180  1  2  6 11  0  1 14 15  0.9979071
6  count_155 10 14 40 56  5 12 81 73  0.9963464
7  count_125  2  1 11  8  2  0 19  8  0.9960246
8  count_105  0  3  8  5  0  0  9  8  0.9916388
9  count_166  6 18  1  0 24 15  0  3  0.9886624
10 count_186  9 58  1  3 24 15  3  4  0.9824930
```

7 Application to Genomic Regions

So far, we have assumed that the count data deals with individual tags from a sequencing machine. If we consider count data derived from grouping together tags relating to genomic regions, for example, in looking at the number of tags that match to a gene in mRNA-Seq, we need to include the region length in the calculations of differential expression. The reasons for this are clear; if a region of length 200 bases has 200 tags in sample A and 400 tags in sample B, this may be good evidence for differential expression. If we saw the same difference in a region 2e6 bases long, this would be much poorer evidence of differential expression. We can specify values for the '@seglens' slot of the 'countData' class to explore such data.

We load a simulated data set consisting of count data on one thousand counts and library sizes for ten libraries. We use the same library sizes as before, but now the first column of 'simSeg' contains the length of the segment.

```
> data(simSeg)
> data(libsizes)
> simSeg[1:10, ]
```

```
      length
 [1,]   6385  150   72   87   61  340  16  28  31   6  45
 [2,]  15078  455  515 1670  120  449  34 133  34  32  63
 [3,]   5577   20    5   11   12   12 106 256  93  47 152
 [4,]  25139  808  529  891 1114 2705 135  89  41  37 312
 [5,]  16423   86   22   36   15   13 344 802 120 462 857
 [6,]  21752 1675  503 1007  193 1033  28   0  10 177 146
 [7,]    434   50   19   29   12   59 196 780  50 214 510
 [8,]  15370   55   45   75   27   77 210 426 137 216 260
 [9,]  43786 2581 1245 2234  940  982 128 644  53 199 257
[10,]  11835   56   26   40   51   55 263 424 441 231 286
```

```
> libsizes
```

```
 [1] 75373 40153 75403 34285 55975 53287 80477 37655 41171 77510
```

As before, the data are simulated such that the first hundred segments show differential expression between the first five libraries and the second five libraries. We thus establish two groups and a replicate structure as before.

```
> replicates <- c(1, 1, 1, 1, 1, 2, 2, 2, 2, 2)
> groups <- list(NDE = c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1), DE = c(1,
+     1, 1, 1, 1, 2, 2, 2, 2, 2))
```

We now combine the count data, library sizes and user-defined groups into a `countData` object as before, but now we additionally specifiy segment lengths by setting the '@seglens' slot.

```
> SD <- new("countData", data = simSeg[, -1], replicates = replicates,
+     libsizes = libsizes, groups = groups, seglens = simSeg[,
+         1])
```

We can also optionally add annotation details into the `@annotation` slot of the `countData` object.

```
> SD@annotation <- data.frame(name = paste("gene", 1:1000, sep = "_"))
```

We can calculate the priors for this countData object as before, for both the Poisson-Gamma method and the Negative Binomial method.

```
> SDP.NBML <- getPriors.NB(SD, samplesize = 1000, estimation = "QL",
+     cl = cl)
> SDP.Pois <- getPriors.Pois(SD, samplesize = 20, cl = cl)
```

We can then calculate the posterior likelihoods as before.

```
> SDPost.Pois <- getLikelihoods.Pois(SDP.Pois, pET = "BIC", cl = cl)
> SDPost.NBML <- getLikelihoods.NB(SDP.NBML, pET = "BIC", cl = cl)
```

.

If we ignore segment length, and merely use the count data, we could acquire posteriors as before.

```
> CSD <- new("countData", data = simSeg[, -1], replicates = replicates,
+     libsizes = libsizes, groups = groups)
> CSD@annotation <- data.frame(name = paste("gene", 1:1000, sep = "_"))
> CSDP.NBML <- getPriors.NB(CSD, samplesize = 1000, estimation = "QL",
+     cl = cl)
> CSDPost.NBML <- getLikelihoods.NB(CSDP.NBML, pET = "BIC", cl = cl)
```

.

```
> CSDP.Pois <- getPriors.Pois(CSD, samplesize = 20, cl = cl)
> CSDPost.Pois <- getLikelihoods.Pois(CSDP.Pois, pET = "BIC", cl = cl)
```

If we look at the false discovery rates (Figure 4), we find that the Negative Binomial method (red) handles data with different segment lengths much better than the Poisson-Gamma method (blue). We also see that, for the Negative Binomial method, ignoring segment lengths can have a small negative effect on the results.

We are also, using the Negative Binomial methods, able to ask if any segments have no true expression by setting `nullData = TRUE`. If the distribution

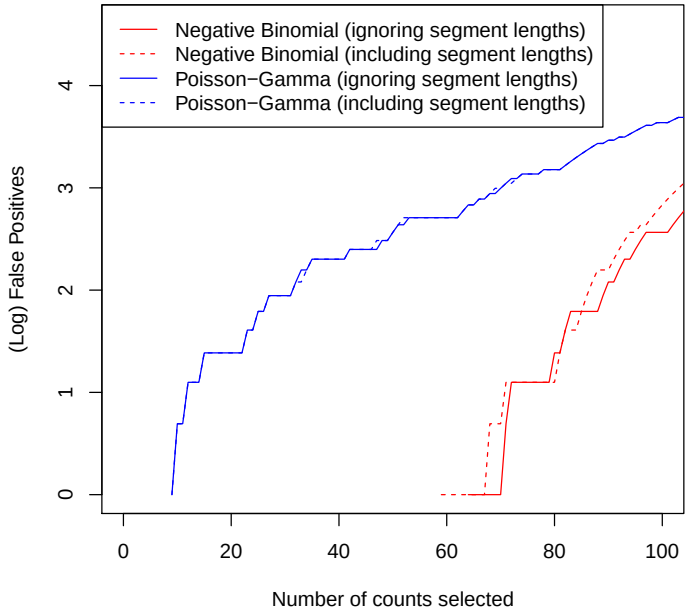

Figure 4: (Log) false positive rates for differentially expressed region discovery in a simulated dataset using both the Negative Binomial and Poisson-Gamma methods, including and ignoring segment length data.

of rates at each segment is bimodal, this approach will be able to differentiate between segments which have no true expression of any kind, and those which have expression (and may be differentially expressed). If we use this option, we implicitly introduce another grouping. The `prs` vector, which defines the prior likelihood of each grouping, should thus sum to less than 1, with the residual going to the implicit group. We can use this method whether or not segment lengths are specified; however, to ignore segment length may have a significant effect on the results.

In the `simSeg` object, the second hundred rows are simulated to show no true expression.

```
> NSDPost.NBML <- getLikelihoods.NB(SDP.NBML, pET = "BIC", nullData = TRUE,
+     bootStraps = 1, cl = cl)
```

.

```
> NCSDPost.NBML <- getLikelihoods.NB(CSDP.NBML, pET = "BIC", nullData = TRUE,
+     bootStraps = 1, cl = cl)
```

.

We can see the top segments for which there is no true expression by setting `group = NULL` in the `topCounts` function. Similarly, we can find true positive rates for by setting `group = NULL` in the `getTPs` function. Figure 5 shows that neglecting the segment lengths causes a substantial difference in false positive rates in the analysis of the data negelecting segment lengths (blue) compared to the analysis of the data including segment length information (red) in the identification of segments for which there is no true expression.

```
> topCounts(NSDPost.NBML, group = NULL)
```

```
       name V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 Likelihood
1  gene_139 10  6 13  5  9 10 13  5 11  19  1.0000000
2  gene_106 32 14 22  5 14 19 21 15 15  19  1.0000000
3  gene_194 17 11 16 10 13  7 27 14  4  22  0.9999999
4  gene_108 12  8 14 11  8  9 13  6  9   7  0.9999999
5  gene_169 24  7  8  8 17 16 16 15 13  13  0.9999999
6  gene_199 29 14 30 19 25  7 29 20 10  26  0.9999998
7  gene_181 14  8  6  4  7  7 10  4  2  11  0.9999998
8  gene_115  9  5 22  7 20  8 16 10 14  26  0.9999997
9  gene_101 16  5  5  6  5  5 12  2  2  10  0.9999997
10 gene_168 11  5 15  6  4 12 16 10 11  19  0.9999997
```

```
> topCounts(NCSDPost.NBML, group = NULL)
```

```
       name V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 Likelihood
1  gene_128  5  1  5  1  3  5  4  1  3   3          1
2  gene_234  6  1  5  2  4  2  3  3  2   8          1
3  gene_218  6  2  4  0  3  2  5  3  4   5          1
4  gene_548  5  1  3  3  5  6  2  2  2   5          1
5  gene_557  4  4 11  3  5  7  6  2  1   7          1
6  gene_735  4  2  6  5  8  3  6  4  7   7          1
7  gene_890  2  3  4  4  5  3 11  3  1   7          1
8  gene_103  8  3  2  3  4  9  8  2  5   4          1
9  gene_193  2  1  1  0  1  3  3  0  1   3          1
10 gene_449  4  3  9  3  3  2  9  2  0   4          1
```

In order to look for segments for which there is no true expression, we assume that the rates of production of tags are bimodally distributed. If this is not the case, then this approach may cause serious problems in the analysis of the data. It is possible to gain some idea of whether the rates are bimodally distributed by examining the priors estimated (by Negative Binomial methods) for a `countData` object by using the function `plotPriors` for some group; ideally, the group which defines non-differentially expressed data. Figure 6 shows that these data are truly bimodal and thus this is a sensible approach.

```
> plotPriors(SDP.NBML, group = 1)
```

References

[1] Thomas J. Hardcastle and Krystyna A. Kelly. *baySeq: Empirical Bayesian Methods For Identifying Differential Expression In Sequence Count Data.* BMC Bioinformatics (2010)

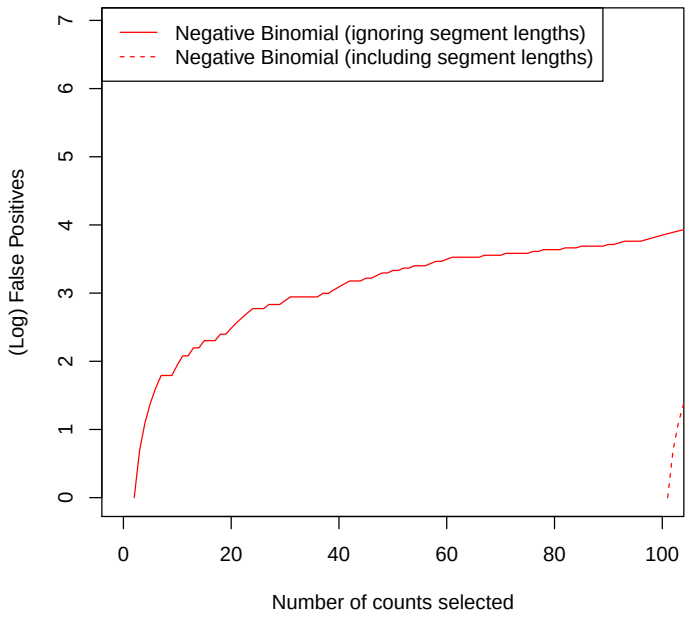

Figure 5: (Log) false positive rates for discovery of regions with no genuine expression in a simulated dataset using the the Negative Binomial approach considering region lengths (solid) and neglecting region lengths (dashed).

[2] Mark Robinson `edgeR`*:' Methods for differential expression in digital gene expression datasets*. Bioconductor.

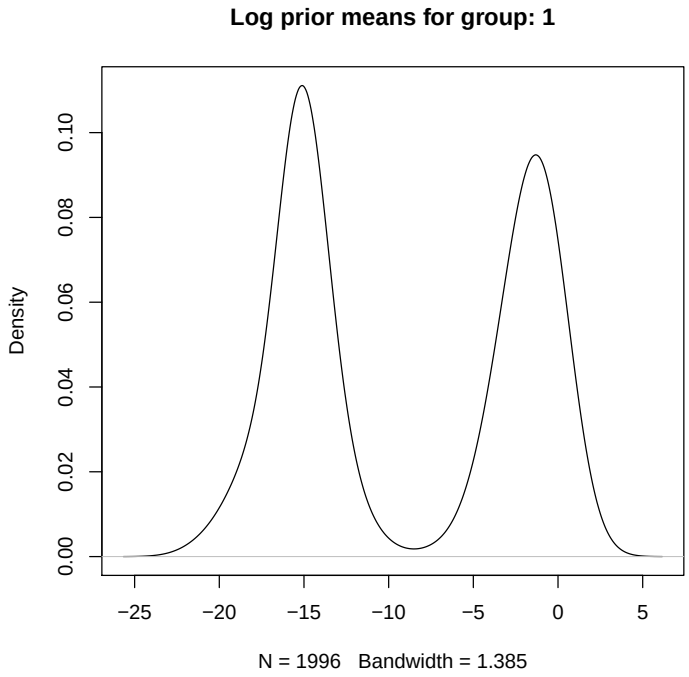

Figure 6: Density of the (log) values estimated for the mean of the data by Negative Binomial methods. The data are bimodally distributed, suggesting that some regions have no true expresssion.